

REMARKS

Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks. Because claims 1-5, 7-12, 15-17, and 22-28 are allowed by the Office, they have not been discussed in this Response.

Patentability Over CORBA in view of Steinman and Hamilton

The Office has asserted a rejection of claims 13, and 18-21, under 35 U.S.C. § 103(a) over “The Common Object Request Broker: Architecture and Specification, CORBA,” Revision 2.0, July 1995 (“CORBA”) in view of Steinman, J., “Incremented State Saving in SPEEDS Using C++,” Proceedings of the 1993 Winter Simulation Conference (“Steinman”) and Coskum, U. S. Patent No. 5,764,958 (“Coskum”). Applicants respectfully traverse.

Claim 13

Claim 13 is generally directed to a method of enhancing scalability of server applications comprising “discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component’s work is complete.”

Specifically, claim 13 recites,

3. Once Amended) In a computer, a method of encapsulating state of processing work for a client by a server application in a component with improved scalability, comprising:
 - encapsulating function code and a processing state for the work in a component;
 - providing a reference through an operating service for a client to call the function code of the component to initiate processing of the work by the component;
 - receiving an indication from the component that the work by the component is complete; and

discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete.
(Emphasis Added).

Thus, the component can indicate discarding its own state upon completion of the work without any indication from the client that the component's work is complete.

The Examiner asserts that the claimed arrangement is obvious in light of a CORBA-Steinman-Coskum combination. Applicants disagree. A CORBA-Coskum-Steinman combination does not teach or suggest “discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete.”

First, the Examiner admits that CORBA fails to teach or suggest the recited arrangement. *See Office Action*, mailed December 20, 2002, page 3, ¶ 2 (“note discussion of claim 21 ... without action by client”). Second, the Examiner does not allege that Steinman teaches or suggests the recited arrangement. Since the references when combined “must teach or suggest all the claim limitations,” Coskum must teach or suggest the recited arrangement.

The Examiner asserts that the following Coskum passages, disclose the recited arrangement,

It is desirable to have a mechanism that can add roles to objects dynamically, depending on the context of the object while limiting the overhead requirements. *Col. 1, ll 56-58.*

...

For example, when a program is started, a person object will only have a student object. Before calling a function requiring teacher characteristics, a teacher role is added to the person object. When the function call returns, the teacher role is deleted from the person object if the role is no longer needed. *Col. 2, ll 14-19.*

...

It will also be apparent to those skilled in the art that when an application program is started requiring only the person object 82, and later requires teacher characteristics, the teacher object 86, may be loaded, and when the function call returns, the teacher object 86, may be deleted if no longer needed. *Col. 4, ll 31-36.*

In summary, Coskum describes a mechanism that adds roles dynamically and deletes roles dynamically as needed. However, these reference along with the references cited by the Office in the previous six Office Actions (mailed November 26, 2000, May 24, 2000, December 19, 2000,

July 30, 2001, September 20, 2001, and June 21, 2002), fail to teach or suggest the claimed language, namely--*who is indicating work is complete?*

Specifically claim 13 recites “discarding the processing state of the component ... *responsive to ... the component indicating completion of the work* ... before receiving any indication from the client that the component’s work is complete.” Coskum does not teach or suggest this. The Examiner continues to ignore the fact that in the cited art, the processing state of the component is not being discarded *responsive to the component’s own indication that work is complete*.

For example, in one passage cited by the Examiner, Coskum states, when “the function call returns, the teacher role is deleted from the person object if the role is no longer needed.” Additionally, “the teacher object 86, may be deleted if no longer needed.” Thus, the teacher object is deleted when it is no longer needed. These statements support the object oriented cannon--objects are created when they are needed, and discarded when they are no longer needed. Since objects are needed to process requests from other objects, they are discarded when no longer needed by requesting objects. Thus, when the teacher object is no longer needed by a requesting object, it is discarded. Coskum adds nothing to the cannon in this regard.

In Cocksum, the teacher object doesn’t know or care whether or not it is still needed. It is agnostic about its own lifespan. It makes no decisions or requests that could in anyway affect its own lifespan. Rather, the teacher object just returns from a function call. In Coskum, requests or decisions about an objects lifespan are made outside the object.

Nowhere does Coskum state that the teacher object is destroyed response to its own indication. In fact, all the present Office Action has done is replace one reference that does not teach or suggest this limitation (Hamilton) with another reference that does not teach or suggest this element (Coskum). Frankly, Coskum adds nothing to the prior Office Action at all. The Examiner continues to ignore that the claim language requires a component being discarded responsive to the component’s own indication that work is complete, and before any indication from the client that work is complete. Without showing this element, no combination can teach or suggest claim 13.

Therefore, a CORBA-Coskum-Steinman combination fails to teach or suggest “discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component’s work is complete.”

For at least this reason claim 13 should be allowed. Such action is respectfully requested.

Claim 14

The Office has indicated that claim 14 is allowable in independent form. Claim 14 has been amended accordingly, and should now be allowable. Such action is respectfully requested.

Claim 18

Amended claim 18 is generally directed to a system service comprising “destroying the processing state of the application component responsive to the indication from the application component that processing by the application component of the work is complete.” Specifically, amended claim 18 recites,

18. (Twice Amended) In a computer, a system service for providing an execution environment for scalable application components, comprising:
code responsive to a request from a client program to create an application component for returning to the client program a reference through the system service to the application component;
code responsive to a call from the client program using the reference for initiating processing of work by the application component, the application component producing a processing state during processing the work;
code for receiving an indication from the application component that processing by the application component of the work is complete; and
code for destroying the processing state of the application component responsive to the indication from the application component that processing by the application component of the work is complete and without action from the client program.
(Emphasis Added).

The Examiner asserts that the recited arrangement is obvious in light of a CORBA-Steinman-Coskum combination. Applicants disagree. A CORBA-Coskum-Steinman combination does not teach or suggest “destroying the processing state of the application component responsive to the indication from the application component that processing by the application component of the work is complete.”

First, the Examiner admits that CORBA fails to teach or suggest the recited arrangement. *See Office Action*, mailed December 20, 2002, page 3, ¶ 2 (“note discussion of claim 21 ... without action by client”). Second, the Examiner does not allege that Steinman teaches or suggests the

recited arrangement. Since the references when combined “must teach or suggest all the claim limitations,” Coskum must teach or suggest the recited arrangement.

The Examiner asserts that the following Coskum passages, disclose the recited arrangement,

It is desirable to have a mechanism that can add roles to objects dynamically, depending on the context of the object while limiting the overhead requirements. *Col. 1, ll 56-58.*

...

For example, when a program is started, a person object will only have a student object. Before calling a function requiring teacher characteristics, a teacher role is added to the person object. When the function call returns, the teacher role is deleted from the person object if the role is no longer needed. *Col. 2, ll 14-19.*

...

It will also be apparent to those skilled in the art that when an application program is started requiring only the person object 82, and later requires teacher characteristics, the teacher object 86, may be loaded, and when the function call returns, the teacher object 86, may be deleted if no longer needed. *Col. 4, ll 31-36.*

In summary, Coskum describes a mechanism that adds roles dynamically and deletes roles dynamically as needed. However, this reference along with the references cited by the Office in the previous six Office Actions (mailed November 26, 2000, May 24, 2000, December 19, 2000, July 30, 2001, September 20, 2001, and June 21, 2002), fail to teach or suggest the claimed language, namely—*destroying responsive to what indication, who is indicating?*

Specifically claim 13 recites “destroying the processing state of the application component responsive to the indication from the application component that processing by the application component of the work is complete.” Coskum does not teach or suggest this. The Examiner continues to ignore the fact that in the cited art, the processing state of the application component is not being discarded *responsive to the component’s own indication that work is complete.*

For example, in one passage cited by the Examiner, Coskum states, when “the function call returns, the teacher role is deleted from the person object if the role is no longer needed.” Additionally, “the teacher object 86, may be deleted if no longer needed.” Thus, the teacher object is deleted when it is no longer needed. These statements support the object oriented cannon--objects are created when they are needed, and discarded when they are no longer needed. Since objects are needed to process requests from other objects, they are discarded when no longer needed by

requesting objects. Thus, when the teacher object is no longer needed by a requesting object, it is discarded. Coskum adds nothing to the cannon in this regard.

In Cocksum, the teacher object doesn't know or care whether or not it is still needed. It is agnostic about its own lifespan. It makes no decisions or requests that could in anyway affect its own lifespan. Rather, the teacher object just returns from a function call. In Coskum, requests or decisions about an objects lifespan are made outside the object.

Nowhere does Coskum state that the teacher object is destroyed response to its own indication. In fact, all the present Office Action has done is replace one reference that does not teach or suggest this limitation (Hamilton) with another reference that does not teach or suggest this element (Coskum). Frankly, Coskum adds nothing to the prior Office Action at all. The Examiner continues to ignore that the claim language requires destroying the processing state of the application component responsive to the indication from the application component that processing by the application component of the work is complete. Without showing this element, no combination can teach or suggest amended claim 18.

Therefore, a CORBA-Coskum-Steinman combination fails to teach or suggest "destroying the processing state of the application component responsive to the indication from the application component that processing by the application component of the work is complete."

For at least this reason claim 18 should be allowed. Such action is respectfully requested.

Claim 19

Claim 19 depends from claim 18. Since claim 19 depends from claim 18, it should be allowed for at least the reasons stated for claim 18. In view of the foregoing discussion claim 18, the merits of the separate patentability of dependent claim 19 not belabored at this time. Claims 19 should be allowed. Such action is respectfully requested.

Claim 20

Claim 20 depends from claim 18. Since claim 20 depend from claim 18, it should be allowed for at least the reasons stated for claim 18. In view of the foregoing discussion claim 18, the merits of the separate patentability of dependent claim 20 is not belabored at this time. Claim 20 should be allowed. Such action is respectfully requested.

Claim 21

Claim 21 is generally directed to a method of enhancing scalability of server applications comprising an “application component having state ... destroying the state by the operating service in response to an indication from the application component without action by the client.”

Specifically, claim 21 recites,

21. (Once amended) In a computer having a main memory, a method of enhancing scalability of server applications, comprising:
executing an application component under control of an operating service, the application component having a state and function code for performing work responsive to method invocations from a client;
maintaining the state in the main memory between the method invocations of the function code by the client in the absence of an indication from the application component that the work is complete;
and
destroying the state by the operating service in response to an indication from the application component without action by the client,
such that the destroyed state is not persistent. (Emphasis Added).

The Examiner asserts that the claimed arrangement is obvious in light of a CORBA-Steinman-Coskum combination. Applicants disagree. A CORBA-Coskum-Steinman combination does not teach or suggest an “application component having state ... destroying the state by the operating service in response to an indication from the application component without action by the client.”

First, the Examiner admits that CORBA fails to teach or suggest the recited arrangement. *See Office Action*, mailed December 20, 2002, page 3, ¶ 2 (“note discussion of claim 21 ... without action by client”). Second, the Examiner does not allege that Steinman teaches or suggests the recited arrangement. Since the references when combined “must teach or suggest all the claim limitations,” Coskum must teach or suggest the recited arrangement.

The Examiner asserts that the following Coskum passages, disclose the recited arrangement,

It is desirable to have a mechanism that can add roles to objects dynamically, depending on the context of the object while limiting the overhead requirements. *Col. 1, ll 56-58.*

...

For example, when a program is started, a person object will only have a student object. Before calling a function requiring teacher characteristics, a teacher role is added to the person object. When

the function call returns, the teacher role is deleted from the person object if the role is no longer needed. *Col. 2, ll 14-19.*

...

It will also be apparent to those skilled in the art that when an application program is started requiring only the person object 82, and later requires teacher characteristics, the teacher object 86, may be loaded, and when the function call returns, the teacher object 86, may be deleted if no longer needed. *Col. 4, ll 31-36.*

In summary, Coskum describes a mechanism that adds roles dynamically and deletes roles dynamically as needed. However, these reference along with the references cited by the Office in the previous six Office Actions (mailed November 26, 2000, May 24, 2000, December 19, 2000, July 30, 2001, September 20, 2001, and June 21, 2002), fail to teach or suggest the claimed language, namely—*whose state is being destroyed responsive to whose indication?*

Specifically claim 13 recites an “application component having state ... destroying the state by the operating service in response to an indication from the application component without action by the client.” Coskum does not teach or suggest this. The Examiner continues to ignore the fact that in the cited art, the state of the application component is not being destroyed ***responsive to the application component’s own indication.***

For example, in one passage cited by the Examiner, Coskum states, when “the function call returns, the teacher role is deleted from the person object if the role is no longer needed.” Additionally, “the teacher object 86, may be deleted if no longer needed.” Thus, the teacher object is deleted when it is no longer needed. These statements support the object oriented cannon--objects are created when they are needed, and discarded when they are no longer needed. Since objects are needed to process requests from other objects, they are discarded when no longer needed by requesting objects. Thus, when the teacher object is no longer needed by a requesting object, it is discarded. Coskum adds nothing to the cannon in this regard.

In Cocksum, the teacher object doesn’t know or care whether or not it is still needed. It is agnostic about its own lifespan. It makes no decisions or requests that could in anyway affect its own lifespan. Rather, the teacher object just returns from a function call. In Coskum, requests or decisions about an objects lifespan are made outside the object.

Nowhere does Coskum state that the teacher object is destroyed response to its own indication. In fact, all the present Office Action has done is replace one reference that does not teach

or suggest this limitation (Hamilton) with another reference that does not teach or suggest this element (Coskum). Frankly, Coskum adds nothing to the prior Office Action at all. The Examiner continues to ignore that the claim language requires an “application component having state ... destroying the state by the operating service in response to an indication from the application component without action by the client.” Without showing this element, no combination can teach or suggest claim 21.

Therefore, a CORBA-Coskum-Steinman combination fails to teach or suggest an “application component having state ... destroying the state by the operating service in response to an indication from the application component without action by the client.”

For at least this reason claim 21 should be allowed. Such action is respectfully requested.

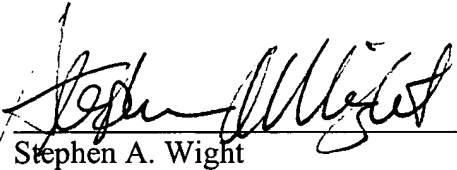
CONCLUSION

The claims in their present form should now be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

By


Stephen A. Wight

Registration No. 37,759

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 226-7391
Facsimile: (503) 228-9446

(80683.1USORG)

**Marked-up Version of Amended Claims
Pursuant to 37 C.F.R. §§ 1.121(b)-(c)**

1. (Thrice Amended) In a computer having a main memory, a method of enhancing scalability of server applications, comprising:

executing an application component under control of an operating service, the application component having a state and function code for performing work responsive to method invocations from a client;

providing an interface for the operating service to receive an indication from the application component that the work is complete;

maintaining the state in the main memory between the method invocations of the function code by the client in the absence of the indication from the application component that the work is complete; and

destroying the state of the application component in response to the indication from the application component to the operating service at the provided interface that the work is complete and without action by the client.

2. (Once Amended) The method of claim 1 wherein the operating service retains an operating service's reference to the application component and the step of destroying the state comprises releasing the operating service's reference to the application component by the operating service while the client retains a client's reference to the application component.

3. (Once Amended) The method of claim 1 wherein the step of destroying the state comprises resetting the state of the application component to the application component's initial post-creation state.

4. (Once Amended) The method of claim 1 wherein said destroying the state is performed by the operating service upon a next return of the application component from the client's call following the indication from the application component that the work is complete.

5. (Unchanged) In a computer, a computer operating environment for scalable, component-based server applications, comprising:

a run-time service for executing an application component in a process, the application component having a state and implementing a set of functions;

an instance creation service operative, responsive to a request of a client, to return a reference to the application component through the run-time service to the client, whereby the client calls functions of the application component indirectly through the run-time service using the reference to initiate work by the application component; and

the run-time service being operative, responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state on the application component returning from a call by the client without action by the client.

7. (Unchanged) The computer operating environment of claim 5 wherein the application component initiates the indication before returning from the call by the client, whereby the application component's state is destroyed immediately on return from the client's call without further action by the client.

8. (Once Amended) In a computer, a computer operating environment for scalable, component-based server applications, comprising:

a run-time service for executing an application component in a process, the application component having a state and implementing a set of functions;

an instance creation service operative, responsive to a request of a client, to return a reference to the application component through the run-time service to the client, whereby the client calls functions of the application component indirectly through the run-time service using the reference to initiate work by the application component;

the run-time service being operative, responsive to an indication from the application component that the application component has completed the work for the client, to destroy the application component's state on the application component returning from a call by the client without action by the client; and

a component context associated by the run-time service with the application component and providing an interface having a member function that the application component calls to initiate the indication.

9. (Unchanged) The computer operating environment of claim 8 wherein the application component performs the work within a transaction and wherein calling the member function of the component context causes the transaction to abort.

10. (Unchanged) The computer operating environment of claim 8 wherein the application component performs the work within a transaction and wherein calling the member function of the component context permits the transaction to commit.

11. (Unchanged) The computer operating environment of claim 5 wherein the run-time service holds a reference to an instance of the application component, and destroys the application component's state by releasing the reference to the instance.

12. (Unchanged) The computer operating environment of claim 5 wherein the run-time service destroys the application component's state by resetting the state.

13. (Once Amended) In a computer, a method of encapsulating state of processing work for a client by a server application in a component with improved scalability, comprising:
encapsulating function code and a processing state for the work in a component;
providing a reference through an operating service for a client to call the function code of the component to initiate processing of the work by the component;
receiving an indication from the component that the work by the component is complete; and
discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete.

14. (Twice Amended) In a computer, a method of encapsulating state of processing work for a client by a server application in a component with improved scalability, comprising:

encapsulating function code and a processing state for the work in a component;
providing a reference through an operating service for a client to call the function code of the component to initiate processing of the work by the component;
receiving an indication from the component that the work by the component is complete;
discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete; and [The method of claim 13 further comprising:]

performing the step of discarding the processing state upon a next return of the component from a call of the client following the indication from the component that the work is complete.

15. (Once Amended) In a computer, a method of encapsulating state of processing work for a client by a server application in a component with improved scalability, comprising:

encapsulating function code and a processing state for the work in a component;
providing a reference through an operating service for a client to call the function code of the component to initiate processing of the work by the component;
receiving an indication from the component that the work by the component is complete; and
discarding the processing state of the component responsive to the component indicating completion of the work before receiving any indication from the client that the component's work is complete;

wherein the step of receiving the indication includes providing a context object containing data representing a context of the component and having an integration interface for receiving a call of the component to indicate that the work by the component is complete.

16. (Unchanged) The method of claim 15 wherein the call of the component to the integration interface of the context object further indicates that a transaction encompassing the work is to be committed.

17. (Unchanged) The method of claim 15 wherein the call of the component to the integration interface of the context object further indicates that a transaction encompassing the work is to be aborted.

18. (Twice Amended) In a computer, a system service for providing an execution environment for scalable application components, comprising:

code responsive to a request from a client program to create an application component for returning to the client program a reference through the system service to the application component;

code responsive to a call from the client program using the reference for initiating processing of work by the application component, the application component producing a processing state during processing the work;

code for receiving an indication from the application component that processing by the application component of the work is complete; and

code for destroying the processing state of the application component **responsive to the indication from the application component that processing by the application component of the work is complete and** without action from the client program.

19. (Unchanged) The system service of claim 18 further comprising:

code for producing an instance of the application component and retaining a reference to the instance, the instance containing the processing state; and

wherein the code for destroying the processing state comprises code for releasing the reference to the instance without action from the client program to thereby cause the processing state to be destroyed.

20. (Unchanged) The system service of claim 18 wherein the code for destroying the processing state comprises:

code for resetting the processing state to an initial state of the application component.

21. (Once Amended) In a computer having a main memory, a method of enhancing scalability of server applications, comprising:

executing an application component under control of an operating service, the application component having a state and function code for performing work responsive to method invocations from a client;

maintaining the state in the main memory between the method invocations of the function code by the client in the absence of an indication from the application component that the work is complete; and

destroying the state by the operating service in response to an indication from the application component without action by the client, such that the destroyed state is not persistent.

22. (Unchanged) In a computer having a main memory, a method of enhancing scalability of server applications, comprising:

executing an application component under control of an operating service, the application component having a component data state and function code for performing work on a data resource responsive to method invocations from a client, the component's work on the data resource having a work data state, the component data state initially having an initial post-creation state upon the component's creation;

providing an interface for the operating service to receive an indication from the application component that the work is complete;

maintaining the component data state in the main memory between the method invocations of the function code by the client in the absence of the indication from the application component that the work is complete; and

in response to the indication and without client action, destroying the component data state by the operating service upon a next return of the application component from a method invocation of the client, while persistently maintaining the work data state.

23. (Unchanged) The method of claim 22 wherein the step of destroying the component data state comprises resetting the component data state to the initial post-creation state.

24. (Once Amended) A computer readable program code-carrying media having software program code encoded thereon, the software program code for executing on a server

computer and implementing a method of enhancing scalability of server applications, the method comprising:

executing an application component under control of an operating service, the application component having a component data state and function code for performing work on a data resource responsive to method invocations from a client, the component's work on the data resource having a work data state, the component data state initially having an initial post-creation state upon the component's creation;

providing an interface for the operating service to receive an indication from the application component that the work is complete;

maintaining the component data state in the main memory between the method invocations of the function code by the client in the absence of the indication from the application component that the work is complete; and

in response to the indication and without client action, destroying the component data state by the operating service upon a next return of the application component from a method invocation of the client, while persistently maintaining the work data state.

25. (Unchanged) In a computer having a main memory, an application component having a state and function code for performing work responsive to method invocations from a client, an improved method of enhancing scalability of server applications, comprising:

providing an interface for the operating service to receive an indication from the application component that the work is complete;

maintaining the state in the main memory between the method invocations of the function code by the client in the absence of the indication from the application component that the work is complete; and

destroying the state of the application component in response to the indication from the application component to the operating service at the provided interface that the work is complete and without an indication from the client allowing the state of the application component to be destroyed.

26. (Unchanged) The method of claim 25, further comprising:

subsequent to the destroying step, restoring the state of the application component in main memory upon receiving a method invocation from the client.

27. (Unchanged) The method of claim 26, further comprising:
subsequent to the restoring step, destroying the state of the application component in response to the indication from the application component to the operating service at the provided interface that the work is complete and without an indication from the client allowing the state of the application component to be destroyed.

28. (Unchanged) The method of claim 25, wherein the destroying step does not include notifying the client that the state of the application component is destroyed.